Lab Assignment 1 - Lab 1

Problem Statement

Write a python program to create a neuron and predict its output using the threshold activation function.

Theory

Neurons

A neuron is the building block of ANN, inspired from biological neurons in brain.

It takes multiple input, applies weights, computes a sum, and passes result through activation function to determine the output.

It consists of:

- Inputs: (x_1, x_2, … x_n)
- Weights (w_1, w_2, ... W_n)
- Summation function ($S = \sum(w_i \times x_i)$)
- Activation function

Threshold Activation Function

Threshold Activation Function is the simplest activation function that decides whether the neuron should fire or not based on the computed sum.

$$f(s) = \begin{cases} 1, & if\ S \geq 0 \\ 0, & if\ S < 0 \end{cases}$$

This function is useful for binary classification problems (Yes/No, True/False, On/Off decisions).

Program

```
class Neuron:

    def __init__(self, weights, threshold):

        self.weights = weights

        self.threshold = threshold


    def activate(self, inputs):

        weighted_sum = sum(w * i for w, i in zip(self.weights, inputs))
```

```
        return 1 if weighted_sum >= self.threshold else 0

weights = list(map(float, input("Enter weights separated by spaces: ").split()))
threshold = float(input("Enter threshold value: "))
neuron = Neuron(weights, threshold)

inputs = list(map(float, input("Enter inputs separated by spaces: ").split()))
output = neuron.activate(inputs)
print("Neuron Output:", output)
```

Outputs

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn1.py
Enter weights separated by spaces: 0.5 0.7 -0.2
Enter threshold value: 0.6
Enter inputs separated by spaces: 1 0 1
Neuron Output: 0

ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn1.py
Enter weights separated by spaces: 0.5 0.7 -0.2
Enter threshold value: 0.6
Enter inputs separated by spaces: 1 1 0
Neuron Output: 1
```

Calculation

- $S=(0.5\times1)+(0.7\times0)+(-0.2\times1)=0.5+0-0.2=0.3$ (<0.6)
- $S=(0.5\times1)+(0.7\times1)+(-0.2\times0)=0.5+0.7+0=1.2$ (>0.6)

Lab Assignment 1 - Lab 2

Problem Statement

Write a python program to train AND Gate Using Perceptron Learning Algorithm.

Theory

Perceptron:

The perceptron is the simplest form of a neural network used for the classifying linearly separable patterns. Patterns that lie on opposite sides of a hyperplane are called linearly separable patterns.

It is a type of artificial neuron that mimics how biological neurons work. It takes multiple inputs, applies weights, sums them, and then applies an activation function to decide the output.

A single-layer perceptron is useful for linearly separable problems, such as the AND, OR gates, but it cannot solve the XOR problem.

The summing node of the neural model computes a linear combination of the input. The resulting sum is applied to a hard limit activation function.

The neuron produces an output equal to 1 if the hard limiter input is positive, and -1 if it is negative.

The goal of the perceptron is to correctly classify the set of externally applied stimuli $x1$, $x2$, ..., xm into one of two classes, c1 or c2. The decision rule for the classification is to assign the point represented by the inputs $x1$, $x2$, ..., xm to class c1 if the perceptron output y is +1 and to class c2 if it is -1.

Perceptron Learning Algorithm

1. Initialize all weights and bias to zero
2. For each training vector s and target t perform steps 3 to 6
3. Set $x_i = s_i \quad for\, i = 1\, to\, n$
4. Compute output using Hard limiter activation function as below
   $$y_{in} = b + \sum_{i=1}^{n} w_i x_i \qquad y = f(y_{in})$$
5. Adapt weights as:
   $w_i = w_i + \alpha(t - y)x_i \quad for\, i = 1\, to\, n$
6. Adapt bias as:
   $b = b + \alpha(t - y)$
7. Test for Stopping Criteria

Program:

```
import numpy as np


class Perceptron:
```

```python
    def __init__(self, learning_rate=0.1, epochs=10):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.random.rand(2)  # Initialize weights randomly
        self.bias = np.random.rand(1)  # Initialize bias randomly

    def activation(self, x):
        return 1 if x >= 0 else -1

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, expected in zip(X, y):
                weighted_sum = np.dot(inputs, self.weights) + self.bias
                output = self.activation(weighted_sum)
                error = expected - output

                # Update weights and bias
                self.weights += self.learning_rate * error * np.array(inputs)
                self.bias += self.learning_rate * error

    def predict(self, inputs):
        weighted_sum = np.dot(inputs, self.weights) + self.bias
        return self.activation(weighted_sum)

# Training data for AND gate
X = np.array([[-1, -1], [-1, 1], [1, -1], [1, 1]])
y = np.array([-1, -1, -1, 1])
```

```
# Train perceptron
perceptron = Perceptron(learning_rate=0.1, epochs=10)
perceptron.train(X, y)


# Test perceptron
print("Trained Perceptron Outputs:")
for inputs in X:
    print(f"Input: {inputs} -> Output: {perceptron.predict(inputs)}")
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn2.py
Trained Perceptron Outputs:
Input: [-1 -1] -> Output: -1
Input: [-1  1] -> Output: -1
Input: [ 1 -1] -> Output: -1
Input: [1 1] -> Output: 1
```

Lab Assignment 1 - Lab 3

Problem Statement

Write a python program to implement Min-Max Scalar.

Theory

Min-Max Scaling (also called Min-Max Normalization) is a feature scaling technique used in machine learning and data preprocessing to rescale numerical data into a specific range, typically [0, 1] or [-1, 1].

For a given value X, the Min-Max scaling formula is:

$$X_{scaled} = \frac{X - X_{min}}{X - X_{max}} \times (R_{max} - R_{min}) + R_{min}$$

Where,

X=original data point

X_{min}, X_{max} = minimum and maximum data in dataset

R_{min}, R_{max} = Desired Range

Limitations of Min-Max Scaling

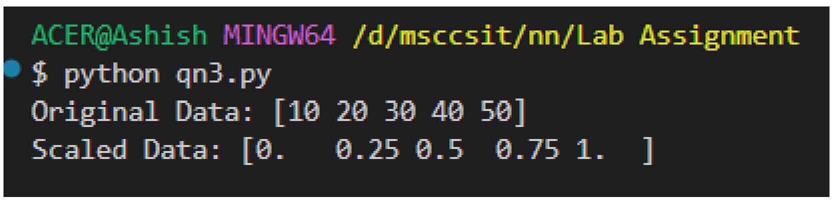
- Sensitive to Outliers: If data has extreme values, Min-Max scaling will compress most values into a narrow range.
- Not Robust: If new data comes in with different min/max values, the scaling needs to be recomputed.

Program

```
import numpy as np


def min_max_scaler(data, feature_range=(0, 1)):
    min_val, max_val = feature_range
    min_data = np.min(data)
    max_data = np.max(data)


    if max_data == min_data:
        return np.zeros_like(data) if min_val == 0 else np.full_like(data, min_val)
```

```
    scaled_data = (data - min_data) / (max_data - min_data) * (max_val - min_val) + min_val
    return scaled_data

data = np.array([10, 20, 30, 40, 50])
scaled_data = min_max_scaler(data, feature_range=(0, 1))
print("Original Data:", data)
print("Scaled Data:", scaled_data)
```

Output



```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn3.py
Original Data: [10 20 30 40 50]
Scaled Data: [0.   0.25 0.5  0.75 1.  ]
```

Lab Assignment 1 - Lab 4

Problem Statement

Write a python program to implement Standard Scalar.

Theory

The Standard Scaler is a preprocessing technique used in machine learning to standardize the features (or variables) of your dataset. The goal of standardization is to transform the data such that each feature has a mean of 0 and a standard deviation of 1, effectively putting all features on the same scale.

For the given value of x, the formula for Standard Scalar is,

$$z = \frac{x - \mu}{\sigma}$$

Where, μ = mean, σ = standard deviation

Program

```
import numpy as np


class StandardScaler:
    def fit(self, X):
        self.mean = np.mean(X, axis=0)
        self.std = np.std(X, axis=0)


    def transform(self, X):
        return (X - self.mean) / self.std


    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)


if __name__ == "__main__":
    data = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
```

```
scaler = StandardScaler()

standardized_data = scaler.fit_transform(data)

print("Original Data:")
print(data)

print("\nStandardized Data:")
print(standardized_data)
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn4.py
Original Data:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

Standardized Data:
[[-1.34164079 -1.34164079]
 [-0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079]]
```

Lab Assignment 1 - Lab 5

Problem Statement

Write a python program to train perceptron using given training set and predict class for the input (6,82) and (5.3,52)

Height(x_1)	Weight(x_2)	Class(t)
5.9	75	Male
5.8	86	Male
5.2	50	Female
5.4	55	Female
6.1	85	Male
5.5	62	Female

Theory

Let's assume following value for given class labels.

Male = 1

Female = -1

We will use min max scaler to normalize the input value.

We will apply the perceptron learning algorithm to the normalized dataset to train the perceptron.

We will test with the given input data (6, 82) and (5.3, 52)

Program

```
import numpy as np


class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=10):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.random.rand(2)
        self.bias = np.random.rand(1)


    def activation(self, x):
```

```
        return 1 if x >= 0 else -1

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, expected in zip(X, y):
                weighted_sum = np.dot(inputs, self.weights) + self.bias
                output = self.activation(weighted_sum)
                error = expected - output

                # Update weights and bias
                self.weights += self.learning_rate * error * np.array(inputs)
                self.bias += self.learning_rate * error

    def predict(self, inputs):
        weighted_sum = np.dot(inputs, self.weights) + self.bias
        return self.activation(weighted_sum)

def min_max_scaler(data, feature_range=(0, 1)):

    min_val, max_val = feature_range
    min_data = np.min(data, axis=0)  # Find min of each column
    max_data = np.max(data, axis=0)  # Find max of each column

    # Avoid division by zero if min == max for a feature
    if np.any(max_data == min_data):
        return np.zeros_like(data) if min_val == 0 else np.full_like(data, min_val)

    scaled_data = (data - min_data) / (max_data - min_data) * (max_val - min_val) + min_val
```

```
    return scaled_data

# Training data
data = np.array([
    [5.9, 75],
    [5.8, 86],
    [5.2, 50],
    [5.4, 55],
    [6.1, 85],
    [5.5, 62]
])

# Labels: 1 for Male, -1 for Female
labels = np.array([1, 1, -1, -1, 1, -1])

# Normalize the data using Min-Max scaling (for all columns/features)
normalized_data = min_max_scaler(data)



# Initialize and train the Perceptron
perceptron = Perceptron(learning_rate=0.1, epochs=10)
perceptron.train(normalized_data, labels)

# Test inputs
test_inputs = np.array([
    [6, 82],
    [5.3, 52]
])
```

```
# Normalize the test inputs using the same scaling
normalized_test_inputs = min_max_scaler(test_inputs)

# Make predictions for test inputs
predictions = [perceptron.predict(test_input) for test_input in normalized_test_inputs]

# Output predictions
for test_input, prediction in zip(test_inputs, predictions):
    result = "Male" if prediction == 1 else "Female"
    print(f"Input: {test_input} -> Predicted Class: {result}")
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment
$ python qn5.py
Input: [ 6. 82.] -> Predicted Class: Male
Input: [ 5.3 52. ] -> Predicted Class: Female
```

Lab Assignment 2 - Lab 6

Problem Statement

Implement Backpropagation algorithm to train an ANN of configuration 2X2X1 to achieve XOR function. (Use sigmoid and Tanh activation function). You have to implement and online as well as batch gradient descent.

Theory

Backpropagation Algorithm

The backpropagation algorithm is a method used to train neural networks by adjusting their weights to minimize the error between the network's predictions and the actual outputs. It works by measuring the error at the output layer and then propagating this error backward through the network to adjust the weights accordingly.

Steps:

1. Initalize Weights and Biases Randomly
    a. Small random values for all weights (e.g., input to hidden, hidden to output).
    b. Biases often initialized to zero or small random values.
2. Forward Propagation
    a. Compute outputs layer-by-layer.
    b. For input x:
        i. Hidden Layer:
            1. $z_1 = x \times w_1 + b1$ → Linear Transformation
            2. $a_1 = activation(z_1)$ → Activation Function
        ii. Output Layer
            1. $z_2 = a_1 \times w_2 + b2$
            2. $a_2 = activation(z_2)$
    c. a_2 is the final predicted output
3. Compute Error
    a. $Error = Actual\ Output - Predicted\ Output$
4. Backward Propagation: Here we adjust weights by how much they contributed to the error.
    a. Output Layer Gradient
        i. Compute gradient of the loss w.r.t. output
            1. $dz_2 = Error \times activation\_derivative(a2)$
        ii. Compute change needed for weights and biases
            1. $dw_2 = a_1^T \times dz_2$
            2. $db_2 = dz_2$
    b. Hidden Layer Gradient
        i. Backpropagate the error to hidden layer
            1. $dz_1 = (dz_2 \times w_2^T) \times activation\_derivative(a_1)$

ii. Compute update for input-to-hidden weights

1. $dw_1 = x^T \times dz_1$
2. $db_1 = dz_1$

5. Update Weights and Biases
   a. Updates using gradient descent rules
      i. $w = w + learning\_rate \times dw$
      ii. $b = b + learnint\_rate \times db$
6. Repeat for many epochs

Types of Training

1. Online Gradient Descent: Also known as Stochastic Gradient Descent, Update weights after every single input example.
2. Batch Gradient Descent: Calculate updates after seeing the entire dataset once.

Program

```
import numpy as np


#XOR DATASET

X = np.array([[0,0], [0,1], [1,0], [1,1]])

Y = np.array([[0], [1], [1], [0]])


#ACTIVATION FUNCTION

def sigmoid(x):

   return 1 / (1 + np.exp(-x))


def sigmoid_derivative(x):

   return x * (1 - x)


def tanh(x):

   return np.tanh(x)


def tanh_derivative(x):
```

```
    return 1 - np.tanh(x)**2

#ALGORITHM
def train_xor(X, Y, activation="sigmoid", epochs=10000, learning_rate=0.1, batch_mode=False):
    #Initialize weights and biases
    np.random.seed(42)
    w1 = np.random.rand(2, 2) #INPUT TO HIDDEN
    b1 = np.zeros((1, 2))
    w2 = np.random.rand(2, 1) #HIDDEN TO OUTPUT
    b2 = np.zeros((1, 1))

    #Choose activation
    if activation == "sigmoid":
        activation_function = sigmoid
        activation_derivative = sigmoid_derivative
    elif activation == "tanh":
        activation_function = tanh
        activation_derivative = tanh_derivative
    else:
        raise ValueError("Invalid activation function")

    #Train
    for epoch in range(epochs):
        if batch_mode:
            #forward pass
            z1 = np.dot(X, w1) + b1
            a1 = activation_function(z1)
            z2 = np.dot(a1, w2) + b2
```

```
    a2 = activation_function(z2)

    #backpropagation
    error = Y - a2
    dz2 = error * activation_derivative(a2)
    dw2 = np.dot(a1.T, dz2)
    db2 = np.sum(dz2, axis=0, keepdims=True)

    dz1 = np.dot(dz2, w2.T) * activation_derivative(a1)
    dw1 = np.dot(X.T, dz1)
    db1 = np.sum(dz1, axis=0)

    #updates
    w1 += learning_rate * dw1
    b1 += learning_rate * db1
    w2 += learning_rate * dw2
    b2 += learning_rate * db2
else:
    for i in range(len(X)):
        x = X[i:i+1]
        y = Y[i:i+1]

        #forward pass
        z1 = np.dot(x, w1) + b1
        a1 = activation_function(z1)
        z2 = np.dot(a1, w2) + b2
        a2 = activation_function(z2)
```

```
            #backpropagation
            error = y - a2
            dz2 = error * activation_derivative(a2)
            dw2 = np.dot(a1.T, dz2)
            db2 = dz2

            dz1 = np.dot(dz2, w2.T) * activation_derivative(a1)
            dw1 = np.dot(x.T, dz1)
            db1 = dz1

            #updates
            w1 += learning_rate * dw1
            b1 += learning_rate * db1
            w2 += learning_rate * dw2
            b2 += learning_rate * db2

        if epoch % 1000 == 0:
            loss = np.mean((Y - a2) ** 2)
            print(f"Epoch {epoch}, Loss: {loss: .4f}")

    return w1, b1, w2, b2

#EVALUATION
def evaluate_xor(X, w1, b1, w2, b2, activation="sigmoid"):
    if activation == "sigmoid":
        activation_function = sigmoid
    elif activation == "tanh":
        activation_function = tanh
```

```
    a1 = activation_function(np.dot(X, w1) + b1)
    a2 = activation_function(np.dot(a1, w2) + b2)
    return a2

# MAIN
def main():
    print("Train XOR ANN (2x2x1)")
    activation = input("Choose activation function (sigmoid/tanh): ").strip().lower()
    mode = input("Choose training mode (batch/online): ").strip().lower()
    epochs = int(input("Number of training epochs [default: 10000]: ") or 10000)
    lr = float(input("Learning rate [default: 0.1]: ") or 0.1)

    batch_mode = True if mode == "batch" else False

    print("\nTraining started...\n")
    w1, b1, w2, b2 = train_xor(X, Y, activation=activation, epochs=epochs, learning_rate=lr,
batch_mode=batch_mode)

    preds = evaluate_xor(X, w1, b1, w2, b2, activation=activation)
    print("Input\tExpected\tPredicted")
    for i in range(len(X)):
        x1, x2 = X[i]
        expected = Y[i][0]
        predicted = np.round(preds[i][0])
        print(f"{x1} {x2}\t   {expected}\t\t   {predicted}")

if __name__ == "__main__":
    main()
```

Output

1. With Online Gradient Descent
   a. Tanh Activation Function
      i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): tanh
Choose training mode (batch/online): online
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Epoch 0, Loss:  0.2500
Epoch 1000, Loss:  0.4908
Epoch 2000, Loss:  0.4974
Epoch 3000, Loss:  0.4986
Epoch 4000, Loss:  0.4990
Epoch 5000, Loss:  0.4993
Epoch 6000, Loss:  0.4994
Epoch 7000, Loss:  0.4995
Epoch 8000, Loss:  0.4996
Epoch 9000, Loss:  0.4996
Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               -0.0
```

   b. Sigmoid Activation Function
      i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): sigmoid
Choose training mode (batch/online): online
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Epoch 0, Loss:  0.2544
Epoch 1000, Loss:  0.2509
Epoch 2000, Loss:  0.2543
Epoch 3000, Loss:  0.2589
Epoch 4000, Loss:  0.2721
Epoch 5000, Loss:  0.3630
Epoch 6000, Loss:  0.4028
Epoch 7000, Loss:  0.4224
Epoch 8000, Loss:  0.4342
Epoch 9000, Loss:  0.4421
Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

2. With Batch Gradient Descent
    a. Tanh Activation Function
        i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): tanh
Choose training mode (batch/online): batch
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

    b. Sigmoid Activation Function
        i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): sigmoid
Choose training mode (batch/online): batch
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

Lab Assignment 2 - Lab 7

Problem Statement

Implement Backpropagation algorithm to train an ANN of configuration 3X2X2X1 to achieve majority function with 3-bit data. Output of the network must be 1 when there are two or more 1's in the data. (Use sigmoid and Tanh activation function). You have to implement and online as well as batch gradient descent.

Theory

Majority Function Dataset:

X1	X2	X3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Program:

```
import numpy as np


# Majority Function Dataset
X = np.array([
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 1],
    [1, 0, 0],
```

```
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
])

Y = np.array([
    [0],
    [0],
    [0],
    [1],
    [0],
    [1],
    [1],
    [1]
])

# Activation Functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

def tanh(x):
    return np.tanh(x)

def tanh_derivative(x):
    return 1 - np.tanh(x)**2
```

```
# Training Function
def train_majority(X, Y, activation="sigmoid", epochs=10000, learning_rate=0.1,
batch_mode=False):
    np.random.seed(42)

    w1 = np.random.randn(3, 2)
    b1 = np.zeros((1, 2))

    w2 = np.random.randn(2, 2)
    b2 = np.zeros((1, 2))

    w3 = np.random.randn(2, 1)
    b3 = np.zeros((1, 1))

    if activation == "sigmoid":
        act = sigmoid
        act_derivative = sigmoid_derivative
    elif activation == "tanh":
        act = tanh
        act_derivative = tanh_derivative
    else:
        raise ValueError("Invalid activation function")

    for epoch in range(epochs):
        if batch_mode:
            z1 = np.dot(X, w1) + b1
            a1 = act(z1)
```

```
    z2 = np.dot(a1, w2) + b2
    a2 = act(z2)

    z3 = np.dot(a2, w3) + b3
    a3 = act(z3)

    error = Y - a3
    dz3 = error * act_derivative(a3)
    dw3 = np.dot(a2.T, dz3)
    db3 = np.sum(dz3, axis=0, keepdims=True)

    dz2 = np.dot(dz3, w3.T) * act_derivative(a2)
    dw2 = np.dot(a1.T, dz2)
    db2 = np.sum(dz2, axis=0, keepdims=True)

    dz1 = np.dot(dz2, w2.T) * act_derivative(a1)
    dw1 = np.dot(X.T, dz1)
    db1 = np.sum(dz1, axis=0, keepdims=True)

    w1 += learning_rate * dw1
    b1 += learning_rate * db1
    w2 += learning_rate * dw2
    b2 += learning_rate * db2
    w3 += learning_rate * dw3
    b3 += learning_rate * db3

else:
    for i in range(len(X)):
```

```
x = X[i:i+1]
y = Y[i:i+1]

z1 = np.dot(x, w1) + b1
a1 = act(z1)

z2 = np.dot(a1, w2) + b2
a2 = act(z2)

z3 = np.dot(a2, w3) + b3
a3 = act(z3)

error = y - a3
dz3 = error * act_derivative(a3)
dw3 = np.dot(a2.T, dz3)
db3 = dz3

dz2 = np.dot(dz3, w3.T) * act_derivative(a2)
dw2 = np.dot(a1.T, dz2)
db2 = dz2

dz1 = np.dot(dz2, w2.T) * act_derivative(a1)
dw1 = np.dot(x.T, dz1)
db1 = dz1

w1 += learning_rate * dw1
b1 += learning_rate * db1
w2 += learning_rate * dw2
```

```
            b2 += learning_rate * db2
            w3 += learning_rate * dw3
            b3 += learning_rate * db3

        if epoch % 1000 == 0:
            loss = np.mean((Y - a3) ** 2)
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    return w1, b1, w2, b2, w3, b3

# Evaluation Function
def evaluate_majority(X, w1, b1, w2, b2, w3, b3, activation="sigmoid"):
    if activation == "sigmoid":
        act = sigmoid
    elif activation == "tanh":
        act = tanh

    a1 = act(np.dot(X, w1) + b1)
    a2 = act(np.dot(a1, w2) + b2)
    a3 = act(np.dot(a2, w3) + b3)
    return a3

# User chooses Activation
print("Choose Activation Function:")
print("1. Sigmoid")
print("2. Tanh")
activation_choice = input("Enter 1 or 2: ").strip()
if activation_choice == "1":
```

```
    activation_choice = "sigmoid"
elif activation_choice == "2":
    activation_choice = "tanh"
else:
    print("Invalid choice. Defaulting to Sigmoid.")
    activation_choice = "sigmoid"

# User chooses Training Mode
print("\nChoose Training Mode:")
print("1. Batch Gradient Descent")
print("2. Online Gradient Descent")
mode_choice = input("Enter 1 or 2: ").strip()
if mode_choice == "1":
    batch_mode = True
elif mode_choice == "2":
    batch_mode = False
else:
    print("Invalid choice. Defaulting to Batch mode.")
    batch_mode = True

# Training
print(f"\nTraining using {activation_choice.upper()} activation and {'BATCH' if batch_mode else 'ONLINE'} mode...\n")
w1, b1, w2, b2, w3, b3 = train_majority(X, Y, activation=activation_choice, epochs=10000, learning_rate=0.1, batch_mode=batch_mode)

# Evaluate
preds = evaluate_majority(X, w1, b1, w2, b2, w3, b3, activation=activation_choice)
```

```
# Show final predictions
print("\nFinal Predictions:")
print("Input\t\tExpected\tPredicted")
for i in range(len(X)):
    x1, x2, x3 = X[i]
    expected = Y[i][0]
    predicted = np.round(preds[i][0])
    print(f"{x1} {x2} {x3}\t\t {expected}\t\t {predicted}")


while True:
    user_input = input("\nEnter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: ").strip()
    if user_input.lower() == 'exit':
        print("Goodbye!")
        break
    try:
        bits = list(map(int, user_input.split()))
        if len(bits) != 3 or any(b not in (0, 1) for b in bits):
            print("Invalid input! Please enter exactly three 0 or 1 values.")
            continue
        bits_array = np.array(bits).reshape(1, -1)
        user_pred = evaluate_majority(bits_array, w1, b1, w2, b2, w3, b3, activation=activation_choice)
        user_pred_binary = np.round(user_pred[0][0])
        print(f"Predicted Output: {int(user_pred_binary)}")
    except Exception as e:
        print("Error:", e)
        continue
```

Output

1. With Online Gradient Descent
    1. Sigmoid Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 1

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 2

Training using SIGMOID activation and ONLINE mode...

Epoch 0, Loss: 0.2730
Epoch 1000, Loss: 0.2633
Epoch 2000, Loss: 0.4683
Epoch 3000, Loss: 0.4811
Epoch 4000, Loss: 0.4856
Epoch 5000, Loss: 0.4880
Epoch 6000, Loss: 0.4895
Epoch 7000, Loss: 0.4906
Epoch 8000, Loss: 0.4915
Epoch 9000, Loss: 0.4921

Final Predictions:
Input           Expected        Predicted
0 0 0             0               0.0
0 0 1             0               0.0
0 1 0             0               0.0
0 1 1             1               1.0
1 0 0             0               0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 1 0
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 0 1
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. Tanh Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 2

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 2

Training using TANH activation and ONLINE mode...

Epoch 0, Loss: 0.2912
Epoch 1000, Loss: 0.4995
Epoch 2000, Loss: 0.4998
Epoch 3000, Loss: 0.4998
Epoch 4000, Loss: 0.4999
Epoch 5000, Loss: 0.4999
Epoch 6000, Loss: 0.4999
Epoch 7000, Loss: 0.4999
Epoch 8000, Loss: 0.4999
Epoch 9000, Loss: 0.4999

Final Predictions:
Input           Expected        Predicted
0 0 0             0               -0.0
0 0 1             0               -0.0
0 1 0             0               -0.0
0 1 1             1               1.0
1 0 0             0               -0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. With Batch Gradient Descent
    1. Tanh Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 2

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 1

Training using TANH activation and BATCH mode...

Epoch 0, Loss: 1.1183
Epoch 1000, Loss: 0.0022
Epoch 2000, Loss: 0.0000
Epoch 3000, Loss: 0.0004
Epoch 4000, Loss: 0.0002
Epoch 5000, Loss: 0.0002
Epoch 6000, Loss: 0.0001
Epoch 7000, Loss: 0.0001
Epoch 8000, Loss: 0.0001
Epoch 9000, Loss: 0.0001

Final Predictions:
Input           Expected        Predicted
0 0 0             0               -0.0
0 0 1             0               -0.0
0 1 0             0               -0.0
0 1 1             1               1.0
1 0 0             0               -0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 1 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. Sigmoid Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 1

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 1

Training using SIGMOID activation and BATCH mode...

Epoch 0, Loss: 0.2760
Epoch 1000, Loss: 0.1971
Epoch 2000, Loss: 0.0043
Epoch 3000, Loss: 0.0015
Epoch 4000, Loss: 0.0008
Epoch 5000, Loss: 0.0006
Epoch 6000, Loss: 0.0004
Epoch 7000, Loss: 0.0004
Epoch 8000, Loss: 0.0003
Epoch 9000, Loss: 0.0002

Final Predictions:
Input           Expected        Predicted
0 0 0             0               0.0
0 0 1             0               0.0
0 1 0             0               0.0
0 1 1             1               1.0
1 0 0             0               0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 1 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

Lab Assignment 3 - Lab 8

Problem Statement

Heart Disease Prediction Using MLP

- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode non-numeric input attributes using Label Encoder.
- Construct an MLP with configuration 11x128x64x32x1. Use Adam optimizer and appropriate activation functions and train the model.
- Predict heart disease for test data and display confusion matrix, accuracy, recall, precision and F1-score.

Theory

In this project, we develop a Heart Disease Prediction model using a Multi-Layer Perceptron (MLP), a type of feedforward artificial neural network.

The model aims to predict whether a patient has heart disease based on clinical features such as age, cholesterol levels, blood pressure, and more.

The dataset was first preprocessed by checking and handling missing values, and encoding categorical attributes using Label Encoding. Feature scaling was applied to standardize numerical inputs, ensuring the model trains efficiently.

An MLP model with architecture 11x128x64x32x1 was built using TensorFlow and Keras libraries. The network uses ReLU activation functions in hidden layers and Sigmoid activation in the output layer to perform binary classification. The model is optimized using the Adam optimizer and trained with binary cross-entropy loss.

After training, the model's performance was evaluated using metrics such as confusion matrix, accuracy, precision, recall, and F1-score, providing a comprehensive understanding of its prediction capabilities.

Program

The following link contains the step-by-step code snippet from google collab notebook.

https://colab.research.google.com/drive/1IyI3P3cHUKvgSNNPCC281bashSzpdVQu?usp=drive_link

Compiled Code

import pandas as pd

```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score,
f1_score

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense


# 1. Load dataset

df = pd.read_csv('lib/reference_dataset/heart.csv')


# 2. Check missing values

print("\nChecking for missing values...")

print(df.isnull().sum())


# Fill missing values if any

df.fillna(df.select_dtypes(include=[np.number]).mean(), inplace=True)



# 3. Display input and output features

print("\nInput features:")

print(df.columns[:-1].tolist())


print("\nOutput feature:")

print(df.columns[-1])


# 4. Encode non-numeric input attributes

le = LabelEncoder()
```

```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])

# 5. Split dataset into X and y
X = df.drop(columns=[df.columns[-1]])  # all except last column
y = df[df.columns[-1]]  # target column

# Normalize the input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# 7. Build MLP Model (11x128x64x32x1)
model = Sequential([
    Dense(128, input_dim=11, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 8. Train the model
print("\nTraining the model...")
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=1)
```

```
# 9. Predict on test data
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# 10. Evaluate performance
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("\n=== Model Evaluation ===")
print("Confusion Matrix:\n", conf_matrix)
print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1-Score: {f1:.4f}")
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn8-heart.py
2025-04-26 17:18:34.411631: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-26 17:18:35.787016: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

Checking for missing values...
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64

Input features:
['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

Output feature:
HeartDisease
D:\msccsit\nn\Lab Assignment\venv\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_c
dels, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-04-26 17:18:38.323015: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compi

Training the model...
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7150 - loss: 0.6004 - val_accuracy: 0.8108 - val_loss: 0.4837
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8796 - loss: 0.3914 - val_accuracy: 0.8108 - val_loss: 0.4337
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8631 - loss: 0.3313 - val_accuracy: 0.8243 - val_loss: 0.4285
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8744 - loss: 0.3168 - val_accuracy: 0.7973 - val_loss: 0.4358
```

```
Epoch 50/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

=== Model Evaluation ===
Confusion Matrix:
 [[66 11]
 [17 90]]
Accuracy: 0.8478
Recall: 0.8411
Precision: 0.8911
F1-Score: 0.8654
(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
```

Lab Assignment 3 – Lab 9

Problem Statement

Iris Prediction using MLP.

- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode output attribute using one hot encoder.
- Shuffle the dataset and then count and display number of tuples in each class.
- Normalize input attributes using standard scalar.
- Split dataset into training/validation/test sets in 70:15:15 ratio.
- Construct an MLP with configuration 4x32x16x8x3. Use Adam optimizer and appropriate activation functions and train the model.
- Predict species of Iris flower for test data and display confusion matrix, weighted avg. accuracy, macro & micro recall, macro & micro precision and macro and micro F1-score.

Theory

The objective of this project is to predict the species of Iris flowers using a Multi-Layer Perceptron (MLP) model. The Iris dataset contains 150 samples of flowers with four input features — sepal length, sepal width, petal length, and petal width — and three output classes: Iris-setosa, Iris-versicolor, and Iris-virginica.

The major steps of the project include:

1. Data Preprocessing
    a. The dataset was checked for missing values (none found).
    b. The target attribute "Species" was one-hot encoded using the OneHotEncoder.
    c. Input features were normalized using StandardScaler to ensure all features are on a similar scale.
    d. The dataset was shuffled and split into training, validation, and test sets in a 70:15:15 ratio.
2. Model Construction:
    a. A Multi-Layer Perceptron (MLP) model was constructed with an architecture of 4 input neurons, followed by hidden layers of 32, 16, and 8 neurons respectively, and finally 3 output neurons (one for each species).
    b. The hidden layers used the ReLU activation function and the output layer used the softmax activation function.
    c. The model was optimized using the Adam optimizer and trained using the categorical cross-entropy loss function.
3. Model Evaluation:
    a. After training, predictions were made on the test set.

b. A confusion matrix and a detailed classification report (precision, recall, f1-score) were generated to evaluate the model's performance.

This project demonstrates the application of a simple feedforward neural network for multi-class classification problems and highlights the importance of proper preprocessing and evaluation techniques in machine learning workflows.

Program:

The following link includes the step-by-step ipynb file for project.

https://colab.research.google.com/drive/1kWh26q5LpyJ-bvNev5k-pFcHmya9t-jV?usp=sharing

Compiled Code:

```
import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report


# Load dataset

df = pd.read_csv('lib/reference_dataset/iris.csv')  # Change the path if needed


# Check missing values

print("\nChecking for missing values...")

print(df.isnull().sum())


# Fill missing values if any

df.fillna(df.select_dtypes(include=['number']).mean(), inplace=True)

# Display input and output features

print("\nInput features:")

print(df.columns[:-1].tolist())
```

```
print("\nOutput feature:")
print(df.columns[-1])

# Display counts for each class
print("Class distribution:\n", df['Species '].value_counts())

# Encode output feature using OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
y = encoder.fit_transform(df[['Species ']])

# Separate input and output
X = df.drop('Species ', axis=1).values

# Normalize input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Shuffle and Split data (70:15:15)
X_temp, X_test, y_temp, y_test = train_test_split(X_scaled, y, test_size=0.15, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=(0.15/0.85), random_state=42, stratify=y_temp)

print(f"Training samples: {len(X_train)}")
print(f"Validation samples: {len(X_val)}")
print(f"Test samples: {len(X_test)}")

# Build MLP model
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(4,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val), verbose=1)

# Evaluate and predict
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Confusion matrix and classification report
print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred_classes))

print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes, digits=4))
```

Output

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn9-iris.py
2025-04-26 17:10:52.470390: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see sligh
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-26 17:10:53.827934: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see sligh
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

Checking for missing values...
SepalLength    0
SepalWidth     0
PetalLength    0
PetalWidth     0
Species        0
dtype: int64

Input features:
['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']

Output feature:
Species
Class distribution:
 Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
Training samples: 104
Validation samples: 23
Test samples: 23
2025-04-26 17:10:57.066703: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimiz
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow wit
Epoch 1/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.4766 - loss: 1.0570 - val_accuracy: 0.5652 - val_loss: 1.0018
Epoch 2/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4138 - loss: 1.0439 - val_accuracy: 0.5217 - val_loss: 0.9732
Epoch 3/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4886 - loss: 0.9909 - val_accuracy: 0.5217 - val_loss: 0.9439
Epoch 4/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4257 - loss: 0.9834 - val_accuracy: 0.5217 - val_loss: 0.9134
Epoch 5/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5336 - loss: 0.9404 - val_accuracy: 0.6087 - val_loss: 0.8818
Epoch 6/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4909 - loss: 0.9052 - val_accuracy: 0.5652 - val_loss: 0.8495
Epoch 7/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4982 - loss: 0.8811 - val_accuracy: 0.6087 - val_loss: 0.8178
Epoch 8/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5486 - loss: 0.8336 - val_accuracy: 0.6522 - val_loss: 0.7861
Epoch 9/50
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9792 - l
Epoch 50/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9692 - l
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step

Confusion Matrix:
[[7 1 0]
 [0 7 1]
 [0 0 7]]

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.8750    0.9333         8
           1     0.8750    0.8750    0.8750         8
           2     0.8750    1.0000    0.9333         7

    accuracy                         0.9130        23
   macro avg     0.9167    0.9167    0.9139        23
weighted avg     0.9185    0.9130    0.9130        23

(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$
```

Lab Assignment

Problem Statement

- Housing Price Prediction
- Check the dataset for missing values and handle, if any.
- Display input and output features of the dataset.
- Encode non-numeric input attributes using label encoder.
- Normalize input and output attributes using standard scalar.
- Split dataset into training/validation/test sets in 70:15:15 ratio.
- Construct an MLP with configuration 12x128x64x32x16x1. Use Adam optimizer and appropriate activation functions and train the model.
- Predict house price for test data.
- Perform inverse transformation of predicted and actual house price.
- Compute and display RMSE, MAE and MAPE.

Theory

The goal of this project is to predict housing prices using a Machine Learning model. The dataset is preprocessed by handling missing values, encoding categorical features using Label Encoding, and normalizing the data with Standard Scaler to ensure features have a mean of 0 and standard deviation of 1.

The dataset is then split into training, validation, and test sets in a 70:15:15 ratio.

A Multi-Layer Perceptron (MLP) with the architecture 12x128x64x32x16x1 is constructed, where hidden layers use the ReLU activation function and the output layer uses a linear activation (since price prediction is a regression task). The Adam optimizer is used to train the model efficiently.

After training, predictions are made on the test set, and an inverse transformation is applied to convert scaled outputs back to actual price values. The model's performance is evaluated using three error metrics: RMSE, MAE, and MAPE.

The formulas for the metrics are:

1. Root Means Square (RMSE):
   a. $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}$
2. Mean Absolute Error (MAE):
   a. $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$
3. $Mean\ Absoulte\ Percentage\ Error\ (MAPE)$:
   a. MAPE = $\frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y_i}}{y_i}\right|$

(Note: MAPE can be undefined or infinite if any true value y_i is zero.)

Program:

The following link includes the step-by-step ipynb file for project.

https://colab.research.google.com/drive/1mdjkwLqPkV5TDgBvn4Nfn5PlzL-b8gSe?usp=sharing

Compiled Code:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import mean_squared_error, mean_absolute_error

import tensorflow as tf


# Load your dataset

df = pd.read_csv('lib/reference_dataset/Housing.csv')


# 1. Check for missing values and handle them

print("Missing Values:\n", df.isnull().sum())

df = df.dropna()  # or you can use df.fillna() for imputation


# 2. Display input and output features

print("\nInput Features:\n", df.columns[:-1])

print("\nOutput Feature:\n", df.columns[-1])


# 3. Encode non-numeric input attributes

label_encoders = {}

for column in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[column] = le.fit_transform(df[column])

    label_encoders[column] = le
```

```
# 4. Normalize input and output attributes
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X = df.iloc[:, :-1].values  # All columns except last
y = df.iloc[:, -1].values.reshape(-1, 1)  # Last column (target)

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

# 5. Split dataset into 70:15:15
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y_scaled, test_size=0.30,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"\nTrain set size: {X_train.shape}")
print(f"Validation set size: {X_val.shape}")
print(f"Test set size: {X_test.shape}")

# 6. Construct the MLP model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),  # input layer
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)  # output layer (no activation)
])
```

```
model.compile(optimizer='adam', loss='mse')

# 7. Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=32,
    verbose=1
)

# 8. Predict house price for test data
y_pred_scaled = model.predict(X_test)

# 9. Perform inverse transformation
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_true = scaler_y.inverse_transform(y_test)

# 10. Compute and display RMSE, MAE and MAPE
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
mae = mean_absolute_error(y_true, y_pred)
epsilon = 1e-8  # Small value to prevent division by zero
mape = np.mean(np.abs((y_true - y_pred) / (y_true + epsilon))) * 100

print(f"\nRMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape:.2f}%")
```

Output:

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn10-Housing.py
2025-04-26 17:44:08.602742: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on.
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDN
2025-04-26 17:44:10.012819: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on.
rom different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDN
Missing Values:
 price              0
area               0
bedrooms           0
bathrooms          0
stories            0
mainroad           0
guestroom          0
basement           0
hotwaterheating    0
airconditioning    0
parking            0
prefarea           0
furnishingstatus   0
dtype: int64

Input Features:
 Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea'],
      dtype='object')

Output Feature:
 furnishingstatus

Train set size: (381, 12)
Validation set size: (82, 12)
Test set size: (82, 12)
2025-04-26 17:44:12.615463: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebui
Epoch 1/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 1.0441 - val_loss: 0.9283
Epoch 2/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.9415 - val_loss: 0.9020
Epoch 3/100
```

```
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1284 - val_loss: 1.3951
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step

RMSE: 1.02
MAE: 0.80
MAPE: 2873846646.69%
(venv)
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
```